Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Дискретная математика и математическая логика

Лабораторная работа №3

Тема: «Минимизация сложных высказываний»

Выполнил: студент группы ИВТ-23-1б

Адаев Даниил Дмитриевич

Проверил: Рустамханова Г. И.

Оглавление

**Цель работы 3**

**Функционал программы 4**

**Заключение 5**

**Тесты 6**

**Код 12**

**Цель работы**

Разработать программу, которая получает на вход бинарный вектор функции и выводит по нему: таблицу истинности, совершенные дизъюнктивную и конъюнктивную нормальные формы, склеивания, импликационную матрицу и минимальные тупиковые формы.

**Функционал программы**

Программа реализована на С++.

Программа представляет собой зацикленное выполнение следующих функций:

vector<int> GetInputVector() – составляет интерфейс для ввода вектора и его сохранения в виде целочисленного массива. Ошибки лишнего символа или неподходящей длины вызывают функцию вновь.

int GetVariables(vector<int> line) – определяет и возвращает необходимое количество переменных для данного вектора степенями двойки.

vector<vector<int>> TruthTable(vector<int> line) – создает, отрисовывает и возвращает двумерный массив таблицы истинности по данному вектору. Номер ряда делится целочисленно на соответствующую переменной степень двойки, затем остаток от деления результата на 2 определяет значение единицы или нуля.

vector<vector<int>> NormalForms(vector<int> line, vector<vector<int>> truth\_table) – выводит совершенные нормальные формы по вектору и таблице истинности. Возвращает массив дизъюнкции. СДНФ определяется по единице в векторе: ряд объединяется конъюнкцией, СКНФ – по нулю в векторе: инвертированный ряд объединяется дизъюнкцией.

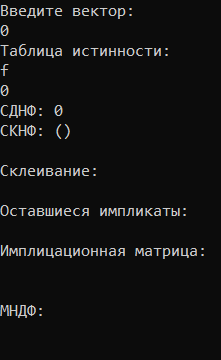
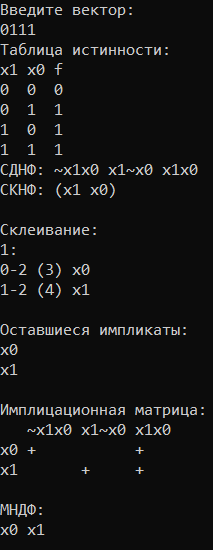
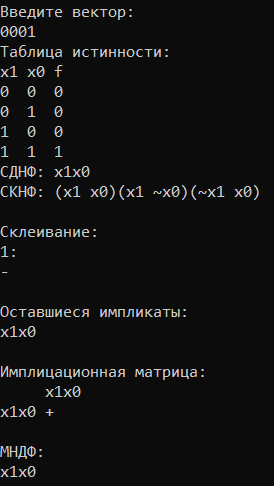
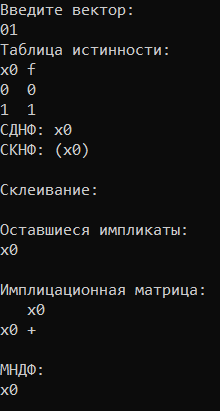
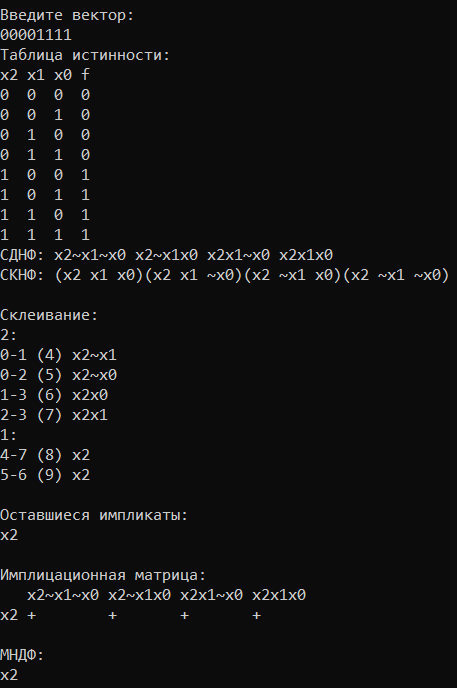
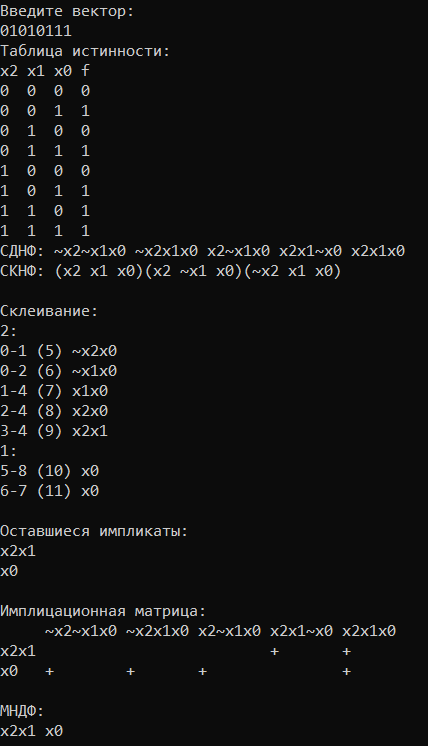
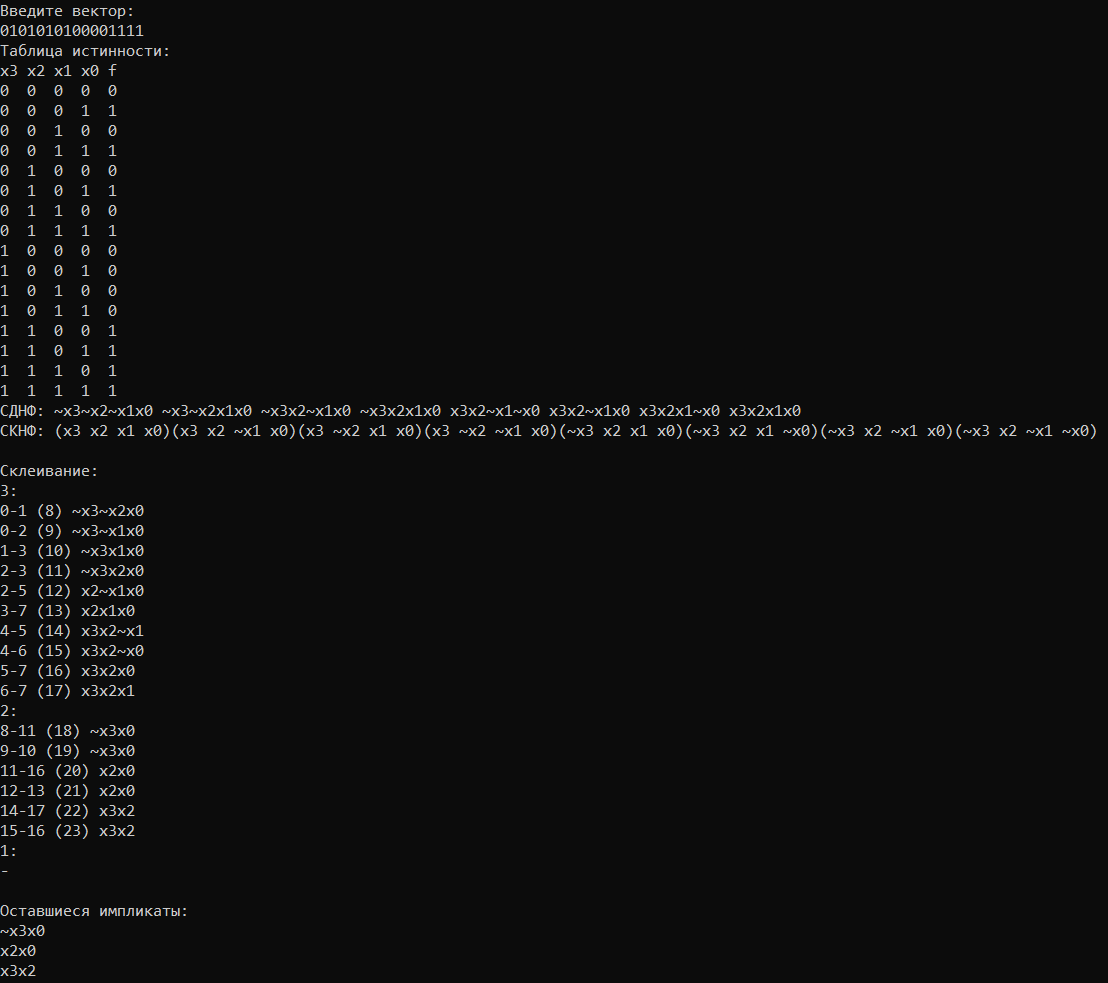
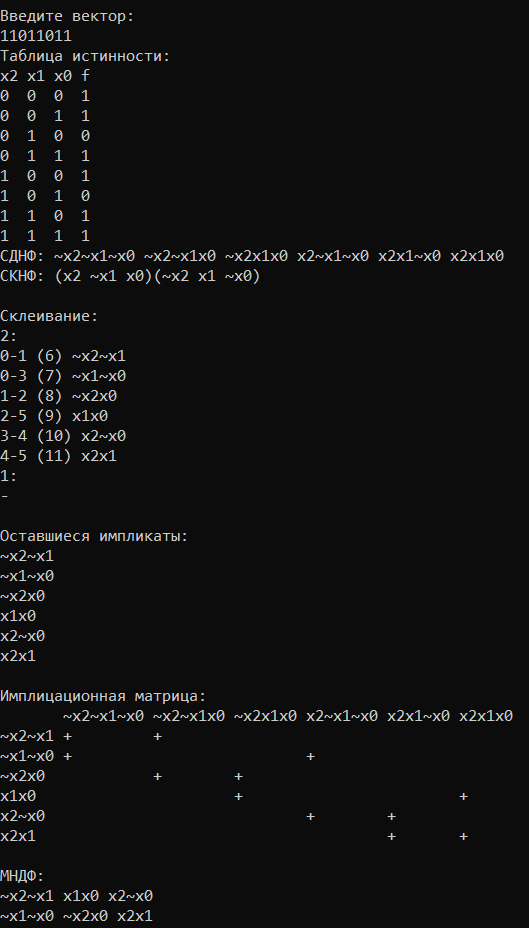
vector<vector<int>> Gluing(vector<vector<int>> disjunction, int variables) – производит склеивание импликат и вывод результатов склеиваний по массиву дизъюнкции и количеству переменных. Возвращает массив импликат. Переменные, исчезнувшие при склеивании, в массиве имеют значение 9 и пропускаются при сравнении.

vector<vector<bool>> ImplicationMatrix(vector<vector<int>> disjunction, vector<vector<int>> implicats, int variables) - создает, отрисовывает и возвращает двумерный массив импликационной матрицы, состоящий из значений true и false.

void MinimalForms(vector<vector<int>> implicats, int variables, vector<vector<bool>> implication\_matrix) – выводит минимальные тупиковые формы по массиву импликат, числу переменных и импликационной матрице. Цикл перебирает незатронутые импликаты, с которых начинает подбирать остальные, подходящие под следующий незатронутый столбец, по максимальному значению очков: +1 за каждую связь с незатронутым столбцом и -1 за каждую связь с уже закрытым столбцом. Полученные формы записываются в трехмерный массив, а затем стираются все, кроме имеющих минимальное число импликат.

**Заключение**

Программа создана успешно.

**Тесты**   
  
  

**Код на C++**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> GetInputVector() {

string input;

cout << "Введите вектор: " << endl;

getline(cin, input);

for (char c : input) {

if (c != '0' && c != '1') {

cout << "Ошибка: Лишний символ в строке. Только 0 и 1 без пробелов." << endl;

return GetInputVector();

}

}

int size = 1;

while (size < input.size()) {

size \*= 2;

}

if (input.size() != size) {

cout << "Ошибка: Неверная длина вектора. Длина должна быть степенью двойки." << endl;

return GetInputVector();

}

vector<int> vectorResult;

for (char c : input) {

vectorResult.push\_back(c - '0');

}

return vectorResult;

}

int GetVariables(vector<int> line){

int variables = 0,

size = line.size();

while (size != 1) {

size /= 2;

variables++;

}

return variables;

}

vector<vector<int>> TruthTable(vector<int> line) {

vector<vector<int>> truth\_table;

int variables = GetVariables(line);

cout << "Таблица истинности:\n";

for (int x = variables - 1; x >= 0; x--) {

vector<int> row;

for (int y = 0; y < line.size(); y++) {

row.push\_back((y / (1 << x)) % 2);

}

truth\_table.push\_back(row);

}

for (int x = variables - 1; x >= 0; x--) {

cout << 'x' << x << ' ';

}

cout << 'f' << endl;

for (int y = 0; y < line.size(); y++) {

for (int x = 0; x < variables; x++) {

cout << truth\_table[x][y] << ' ';

int spaces = x / 10;

for (int i = 0; i < spaces; i++)

{

cout << ' ';

}

cout << ' ';

}

cout << line[y] << endl;

}

return truth\_table;

}

vector<vector<int>> NormalForms(vector<int> line, vector<vector<int>> truth\_table) {

vector<vector<int>> disjunction, conjunction;

int variables = GetVariables(line);

for (int y = 0; y < line.size(); y++) {

vector<int> tmp;

for (int x = 0; x < variables; x++) {

tmp.push\_back(truth\_table[x][y]);

}

if (line[y] == 1) {

disjunction.push\_back(tmp);

}

else {

conjunction.push\_back(tmp);

}

}

cout << "СДНФ: ";

if (disjunction.empty()) {

cout << 0;

}

else {

for (int y = 0; y < disjunction.size(); y++) {

if (y > 0) cout << " ";

for (int x = 0; x < variables; x++) {

if (disjunction[y][x] == 0) cout << '~';

cout << 'x' << variables - 1 - x;

}

}

}

cout << "\nСКНФ: ";

if (conjunction.empty()) {

cout << 1;

}

else {

for (int y = 0; y < conjunction.size(); y++) {

cout << '(';

for (int x = 0; x < variables; x++) {

if (x > 0) cout << " ";

if (conjunction[y][x] == 1) cout << '~';

cout << 'x' << variables - 1 - x;

}

cout << ')';

}

}

cout << endl;

return disjunction;

}

vector<vector<int>> Gluing(vector<vector<int>> disjunction, int variables) {

int current\_implicats\_end = disjunction.size();

int current\_implicats\_begin = 0;

cout << "\nСклеивание:\n";

for (int v = variables - 1; v > 0; v--) {

cout << v << ":\n";

int next\_implicats = 0;

for (int y = current\_implicats\_begin; y < current\_implicats\_end - 1; y++) {

for (int y1 = y + 1; y1 < current\_implicats\_end; y1++) {

vector<int> tmp(variables, 0);

for (int x = 0; x < variables; x++) {

tmp[x] = disjunction[y][x];

}

bool flag0 = true,

flag1 = true;

for (int x = 0; x < variables; x++) {

if (tmp[x] != disjunction[y1][x] and flag0) {

flag0 = false;

tmp[x] = 9;

}else if (tmp[x] != disjunction[y1][x] and !flag0){

flag1 = false;

break;

}

}

if (flag1 and !flag0) {

disjunction.push\_back(tmp);

next\_implicats++;

cout << y << '-' << y1 << " (" << current\_implicats\_end - 1 + next\_implicats << ") ";

for (int x = 0; x < variables; x++) {

if (tmp[x] == 1) {

cout << 'x' << variables - 1 - x;

}

else if (tmp[x] == 0) {

cout << '~' << 'x' << variables - 1 - x;

}

}

cout << endl;

}

}

}

if (next\_implicats == 0)

{

cout << '-' << endl;

}

current\_implicats\_begin = current\_implicats\_end;

current\_implicats\_end += next\_implicats;

}

cout << "\nОставшиеся импликаты:\n";

vector<vector<int>> implicats = disjunction;

int y = disjunction.size() - 1;

while(y > 0) {

for (int y1 = y - 1; y1 >= 0; y1--) {

bool flag = true;

for (int x = 0; x < variables; x++) {

if (disjunction[y][x] != implicats[y1][x] and disjunction[y][x] != 9) {

flag = false;

break;

}

}

if (flag) {

implicats.erase(implicats.begin() + y1);

}

}

if (disjunction.size() > implicats.size()) {

disjunction = implicats;

y = disjunction.size();

}

y--;

}

for (int y = 0; y < implicats.size(); y++) {

for (int x = 0; x < variables; x++) {

if (implicats[y][x] == 1) {

cout << 'x' << variables - 1 - x;

}

else if (implicats[y][x] == 0) {

cout << '~' << 'x' << variables - 1 - x;

}

}

cout << endl;

}

return implicats;

}

vector<vector<bool>> ImplicationMatrix(vector<vector<int>> disjunction, vector<vector<int>> implicats, int variables) {

cout << "\nИмплицационная матрица:\n";

vector<vector<bool>> implication\_matrix;

for (int y = 0; y < implicats.size(); y++) {

vector<bool> tmp;

for (int x = 0; x < disjunction.size(); x++) {

tmp.push\_back(true);

for (int i = 0; i < variables; i++) {

if (implicats[y][i] != disjunction[x][i] and implicats[y][i] != 9) {

tmp[x] = false;

break;

}

}

}

implication\_matrix.push\_back(tmp);

}

string max\_space = "";

for (int y = 0; y < implicats.size(); y++) {

string tmp = "";

for (int x = 0; x < variables; x++) {

if (implicats[y][x] == 0) {

tmp += '~';

}

if (implicats[y][x] != 9) {

tmp += 'x' + to\_string(x);

}

}

while (max\_space.size() < tmp.size()) {

max\_space += ' ';

}

}

max\_space += ' ';

cout << max\_space;

vector<string> spaces;

for (int y = 0; y < disjunction.size(); y++) {

string tmp = "";

for (int x = 0; x < variables; x++) {

if (disjunction[y][x] == 0) {

tmp += '~';

}

if (disjunction[y][x] != 9) {

tmp += 'x' + to\_string(variables - 1 - x);

}

}

cout << tmp;

for (int i = 0; i < tmp.size(); i++) {

tmp[i] = ' ';

}

spaces.push\_back(tmp);

cout << ' ';

}

cout << endl;

for (int y = 0; y < implicats.size(); y++) {

string tmp = "";

for (int x = 0; x < variables; x++) {

if (implicats[y][x] == 0) {

tmp += '~';

}

if (implicats[y][x] != 9) {

tmp += 'x' + to\_string(variables - 1 - x);

}

}

cout << tmp;

for (int i = tmp.size(); i < max\_space.size(); i++) {

cout << ' ';

}

for (int x = 0; x < disjunction.size(); x++) {

if (implication\_matrix[y][x]) {

cout << '+';

}

else {

cout << ' ';

}

cout << spaces[x];

}

cout << endl;

}

return implication\_matrix;

}

void MinimalForms(vector<vector<int>> implicats, int variables, vector<vector<bool>> implication\_matrix) {

cout << "\nМНДФ:\n";

vector<vector<vector<int>>> minimal\_forms;

vector<bool> untouched\_implicats(implicats.size(), true);

for (int ui = 0; ui < implicats.size(); ui++) {

vector<vector<int>> tmp;

if (untouched\_implicats[ui]) {

untouched\_implicats[ui] = false;

tmp.push\_back(implicats[ui]);

vector<bool> untouched\_disjunction(implication\_matrix[0].size(), true);

for (int x = 0; x < implication\_matrix[0].size(); x++) {

if (implication\_matrix[ui][x]) {

untouched\_disjunction[x] = false;

}

}

for (int ud = 0; ud < implication\_matrix[0].size(); ud++) {

if (untouched\_disjunction[ud]) {

int max\_id;

int max\_points;

bool is\_max = true;

for (int y = 0; y < implicats.size(); y++) {

if (implication\_matrix[y][ud]) {

int points = 0;

for (int x = 0; x < implication\_matrix[0].size(); x++) {

if (implication\_matrix[y][x] and untouched\_disjunction[x]) {

points++;

}

else if (implication\_matrix[y][x] and !untouched\_disjunction[x]) {

points--;

}

}

if (is\_max or max\_points < points) {

max\_points = points;

max\_id = y;

}

}

}

for (int x = 0; x < implication\_matrix[0].size(); x++) {

if (implication\_matrix[max\_id][x]) {

untouched\_disjunction[x] = false;

}

}

untouched\_implicats[max\_id] = false;

tmp.push\_back(implicats[max\_id]);

}

}

minimal\_forms.push\_back(tmp);

}

}

bool is\_min = true;

int min;

for (int z = 0; z < minimal\_forms.size(); z++) {

if (is\_min or min > minimal\_forms[z].size()) {

min = minimal\_forms[z].size();

}

}

int size = minimal\_forms.size() - 1;

while (size > 0) {

if (minimal\_forms[size].size() > min) {

minimal\_forms.erase(minimal\_forms.begin() + size);

}

size--;

}

for (int z = 0; z < minimal\_forms.size(); z++) {

for (int y = 0; y < minimal\_forms[z].size(); y++) {

for (int x = 0; x < variables; x++) {

if (minimal\_forms[z][y][x] == 0) {

cout << '~';

}

if (minimal\_forms[z][y][x] != 9) {

cout << 'x' << variables - 1 - x;

}

}

cout << ' ';

}

cout << endl;

}

}

int main()

{

setlocale(LC\_ALL, "RUS");

while(true) {

vector<int> line = GetInputVector();

int variables = GetVariables(line);

vector<vector<int>> truth\_table = TruthTable(line);

vector<vector<int>> disjunction = NormalForms(line, truth\_table);

vector<vector<int>> implicats = Gluing(disjunction, variables);

vector<vector<bool>> implication\_matrix = ImplicationMatrix(disjunction, implicats, variables);

MinimalForms(implicats, variables, implication\_matrix);

cout << "\n\n";

}

return 0;

}